CS 280
Programming Language Concepts

About variables, memory, pointers, and references

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Variables

- Variables have names
- In (most) languages, variables have types
- Variables have some memory associated with them
  - Some languages may have symbolic names that behave like variables but don't necessarily need memory… but for our purposes we can skip that for now
- When the memory gets allocated, and where it gets allocated from, depends on the language and on where the variable is declared
- Some languages let us deal with the memory associated with a variable separately from the variable itself

Memory for variables

- A variable needs enough memory to hold an instance of an object of that type (in other words, the type dictates how much memory is needed)
- `int x;`
  - declares that x is an integer; enough memory is allocated to hold an integer
- `Obj y;`
  - In C++ this means that y is a Obj; enough memory is allocated to hold an Obj
    - A constructor is called if one is provided
  - In Java this means that y is a reference to an Obj; enough memory is allocated to hold a REFERENCE to an Obj
    - By definition Java initializes the reference to null
    - The reference is not the object!

Where, in memory, are the variables?

- This depends on the language
- In most languages, variables declared inside of a function, and variables for function arguments, have memory that is allocated on the stack
- Global variables are placed in the data segment
- Dynamically allocated memory is on the heap, and is assigned using “new”
- The result of “new” must be saved in a variable

Constructors

- Languages like Java and C++ allow the programmer to define a “constructor” for a class
- A constructor is a method whose name is the name of the class
- A constructor will be called immediately after the memory for an instance of the class gets allocated
- Think of it as an initializer

New

- C++ and Java provide a `new` operator
- Using this operator gets memory for a new instance of the type you are using it on (for example a "`new Obj()`")
- Since this allocates memory, "`new Obj()`" causes a constructor for `Obj` to be called if one exists
- In C++ you can overload the `new` operator
- In Java, a reference is returned
- In C++, a pointer is returned

Pointers

- A pointer is a variable that contains a memory address
- Pointers must be explicitly declared
  - In C/C++:
    - `int *ip;` declares ip as a pointer to an int
    - `Obj *op;` declares op as a pointer to an Obj
- Pointers need to be initialized: they must "point to" something:
  - you can assign the value of another pointer to a pointer
  - you can set the pointer to the "address of" something, using the & operator
  - you can assign what is returned from "new" to the pointer
- To get to what the pointer is pointing at, use the * operator to dereference the pointer
- The expression *pointer can be on left or right side of an = sign

References

- In Java, variables that have the type of an object are actually a reference to an instance of the object, not the object itself
- When you use a reference, you are actually using what the reference refers to. You can not see, or change, the memory address
- It would not be wrong to think of a reference as a kind of a pointer: it *does* contain a memory address like a pointer does; however, you cannot see the memory address of the reference, just what it refers to

References

- To initialize a reference in Java, you assign what is returned from "new" to the reference
- References in C++ must be explicitly declared: "`int& x`" is a reference to an integer; the name of the reference is x
- References in C++ must be initialized when declared:

```
int x;
int& xr = x; // without the initialization? compile error
```

- Note: If a parameter to a function is a reference, then it's initialized, at the time that the function is called, to refer to the variable that is passed to the function

So what?

- If I have a pointer to something, or a reference to something, I can follow the pointer or the reference to access, any maybe change, what it points to/refers to
- Pointers and references are smaller than the things they point to
- Why copy big things when you can copy pointers/references?


```
int x,y,z;        // integers
int& xr = x;      // reference to the int x
int *yp;          // a pointer to an int

yp = &y;          // initialize pointer to point to y

xr = 10;          // sets x (what xr refers to) to 10
y = 20;
z = *yp;          // set z to the value of the int
                  // that yp points to

*yp = xr;  // set the int that yp points to
           // equal to the value of x
           //    (what xr refers to)
```

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Careful: this code probably crashes

```
int x;
int *xp;

*xp = 100;
     // what does xp point to??

     // you MUST initialize pointers

     // a pointer MUST point at something
     // in order for you to use it
```


Pointer Initialization

- Dereferencing (using the * operator on) a pointer that has not been initialized is an error
- How the error appears is usually a crash (if you are lucky!): a “core dump” or a “segmentation violation”
- Finding these errors can be a challenge
  - Have your program print messages out so you can isolate where the problem happens
  - If you have a debugger, it can tell you where the error happened

```
class X {
public:
       int x;
};

X p, q;        // instances of X
X& pr = p;     // reference to the instance of X named p
X* qp;         // a pointer to an X
X* rp;

qp = &q;       // initialize pointer to point to q

pr.x = 10;     // sets the x in p (what pr refers to) to 10

// both of these sets the x in what qp points to to 20
(*qp).x = 20;
qp->x = 20;

rp = new X();         // get a new X

*rp = q;       // copy the objects
```


Back To Arrays

- Arrays are a group of items of the same type in contiguous memory
- In C:

```
int x[10]; // an array of 10 integers
// x[i] is an integer
// x is of type int*, whose value is &x[0]
```

- This works in C++ as well
- In C++ you can also:

```
int *x;
x = new int[10];
```

- In both languages you can initialize:

```
int xa[] = {1,2,3,4};
```

Java Arrays

- Java arrays are objects
- Therefore, variables that are declared as arrays are actually references to array objects, so they must have memory assigned
- Declare
  - int[] array;
- Assign memory
  - array = new int[10];
- Every array is an object that has a “length” member, so you know how many items are in it

Passing Arrays As Arguments

- Arrays are not copied to functions; instead a pointer (C/C++) or reference (C++ if you declare it, always a reference in Java) is passed
- Java function arguments can be declared as, for example, String[] args. Since the array has a length method, you know how long the array is (args.length in this example)
- In C/C++ the name of the array is a pointer to the first element of the array.
  - A parameter declared int *ap or int ap[] works the same
  - Note there is no length member

Arrays And Pointers Are Connected

- Accessing a member of an array x[i] involves some calculation to find where the ith element of the array is located
- x[0] is the first element of the array, x[1] is the second, etc
- In actuality, x[i] is a shorthand for *(x+i)
  - The language defines “pointer arithmetic”
  - The semantics of pointer + integer is defined to actually be pointer + (integer * the size of what pointer points at)
  - chararray[0] is the first char (this is why programmers start counting from 0)
  - chararray[3] is thus the 4nd char, intarray[4] is the 5th int
  - x[i] == *(x+i) == *(i+x) == i[x] !!!!!!